PETSc with libCEED - Performance Portable Matrix-Free Operators

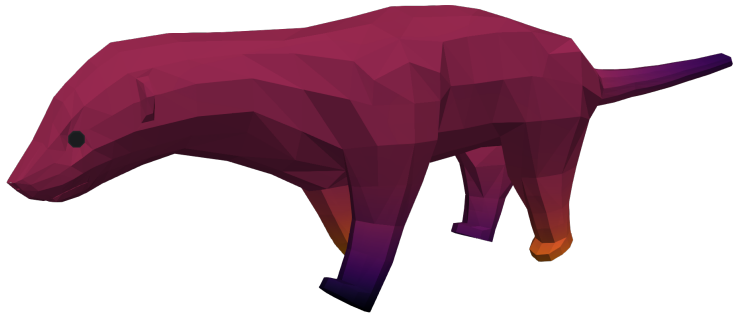
Jeremy L Thompson

University of Colorado Boulder

jeremy@jeremylt.org

24 May 2024

libCEED Examples Team



libCEED Repo: https://github.com/CEED/libCEED
Ratel Repo: https://gitlab.com/micromorph/ratel

Developers: Zach Atkins, Jed Brown, Leila Ghaffari, Kenneth Jansen
Rezgar Shakeri, James Wright, Jeremy L Thompson

The authors acknowledge support by the Department of Energy, National Nuclear Security Administration, Predictive Science Academic Alliance Program (PSAAP) under Award Number DE-NA0003962.

Overview

ECP Roots

- libCEED + PETSc projects follow from ECP CEED work
- libCEED provides high-performance operator evaluation
- libCEED provides CUDA, ROCm, and SYCL support
- PETSc provides linear/non-linear solvers and time steppers

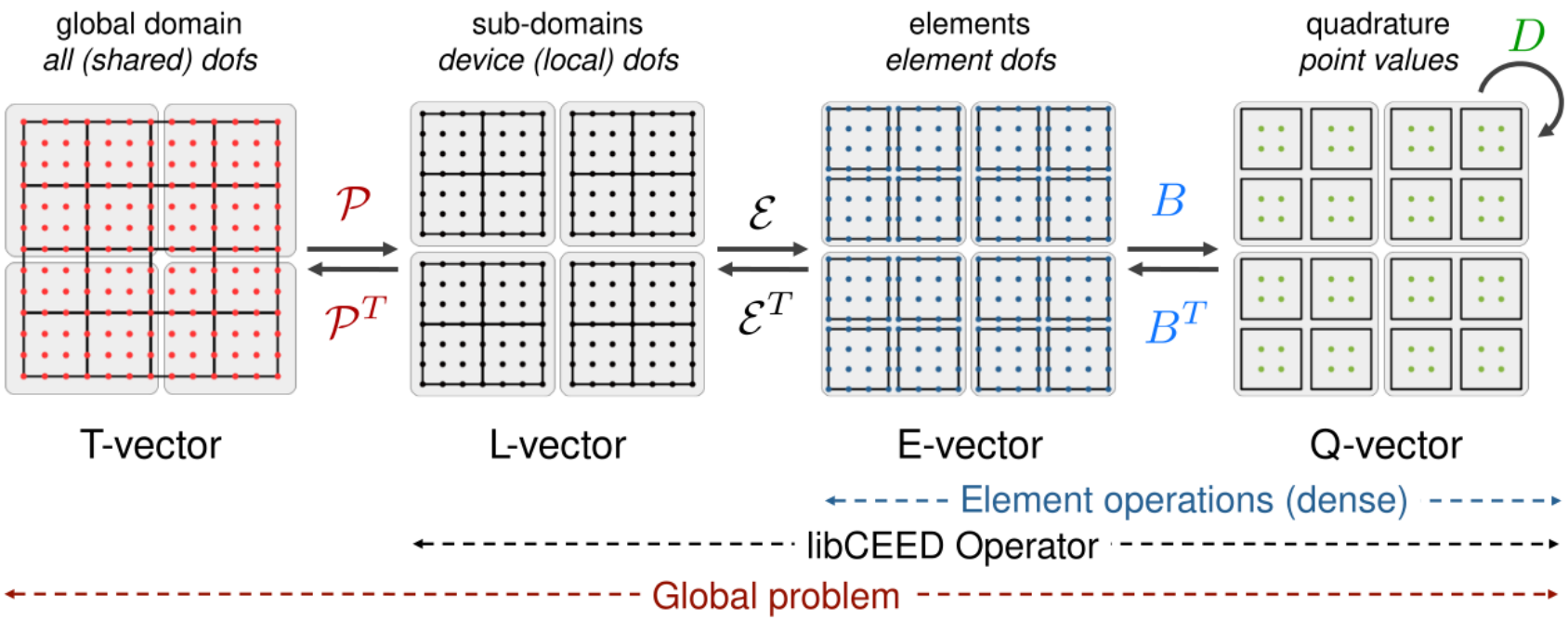
libCEED Projects

Several projects built using libCEED

- Ratel - solid mechanics FEM and MPM (PSAAP)
- HONEE - fluid dynamics FEM & differential filtering (PHASTA)
- MFEM - various applications, libCEED integrators (LLNL)
- Palace - Electromagnetics FEM with MFEM + libCEED (Amazon)
- RDycore - River dynamical core with PETSc + libCEED (SciDAC)

Matrix-Free Operators from libCEED

$$A = \mathcal{P}^T \mathcal{E}^T B^T D B \mathcal{E} \mathcal{P}$$



libCEED provides arbitrary order matrix-free operator evaluation

Neo Hookean QFunction

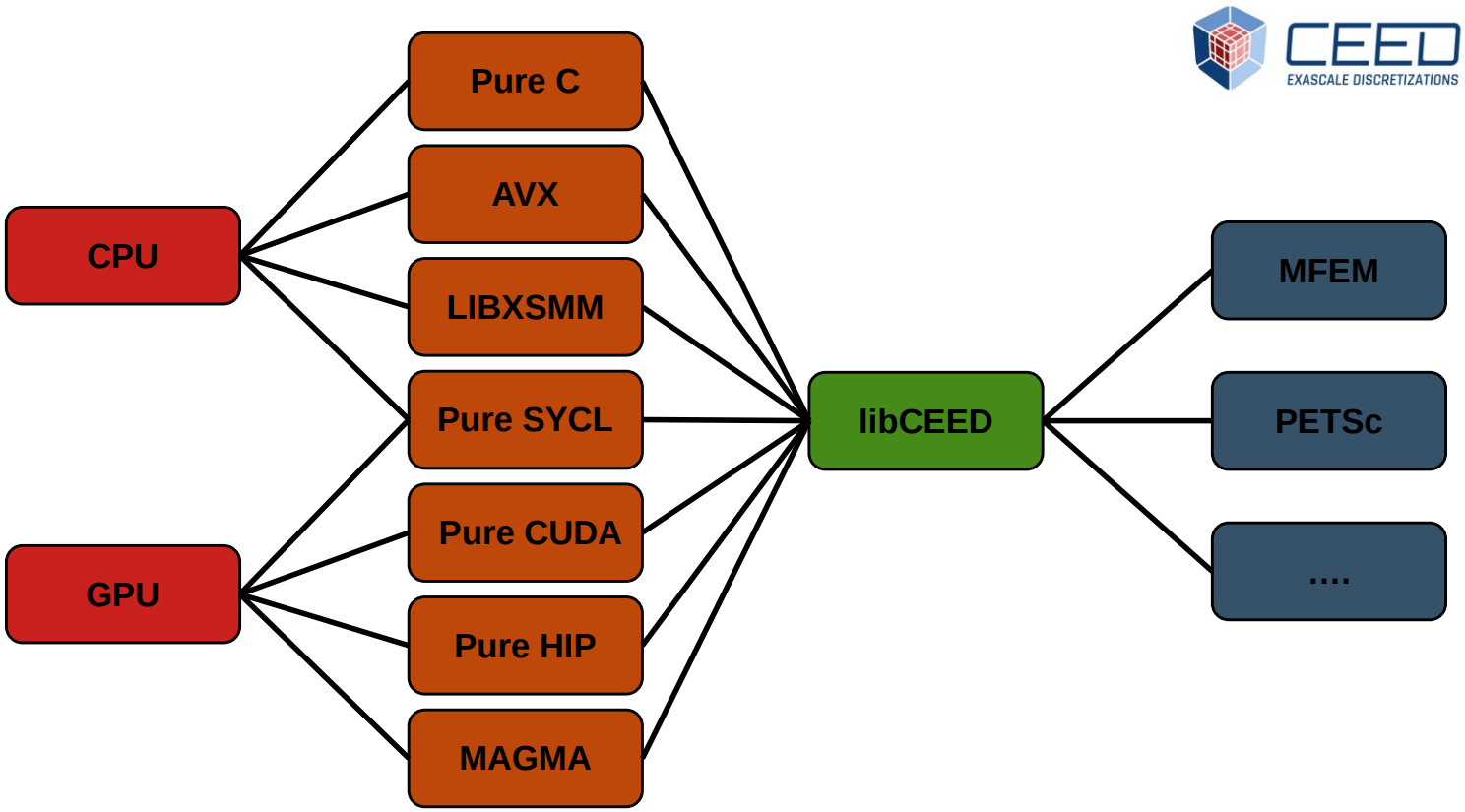
```c
static int ElasticityResidual_NeoHookean(void *ctx, CeedInt Q,
                                         const CeedScalar *const *in, CeedScalar *const *out) {
  // Inputs
  const CeedScalar(*q_data)[CEED_Q_VLA] = (const CeedScalar(*)[CEED_Q_VLA])in[0];
  const CeedScalar(*ug)[CEED_Q_VLA][Q]  = (const CeedScalar(*)[3][CEED_Q_VLA])in[1];

  // Outputs
  CeedScalar(*grad_u)[3][CEED_Q_VLA] = (CeedScalar(*)[3][CEED_Q_VLA])out[0];

  // Context
  const RatelNeoHookeanParams *context = (RatelNeoHookeanParams *)ctx;
  const CeedScalar lambda = context->lambda;
  const CeedScalar two_mu = context->two_mu;

  // Quadrature Point Loop
  CeedPragmaSIMD for (CeedInt i = 0; i < Q; i++) {
    ...
    // Compute J - 1
    const CeedScalar Jm1 = RatelMatDetAM1(temp_grad_u);
    // Compute E, C^{-1}, S
    RatelGreenLagrangeStrain(temp_grad_u, E_sym);
    RatelCInverse(E_sym, Jm1, C_inv_sym);
    RatelSecondKirchhoffStress(lambda, two_mu, Jm1, C_inv_sym, E_sym, S_sym);
    RatelSymmetricMatUnpack(S_sym, S);
    // Compute the First Piola-Kirchhoff f1 = P = F*S
    RatelMatMatMult(1.0, F, S, f1);
    ...
  }
  return CEED_ERROR_SUCCESS;
}
```

Performance Portability from libCEED



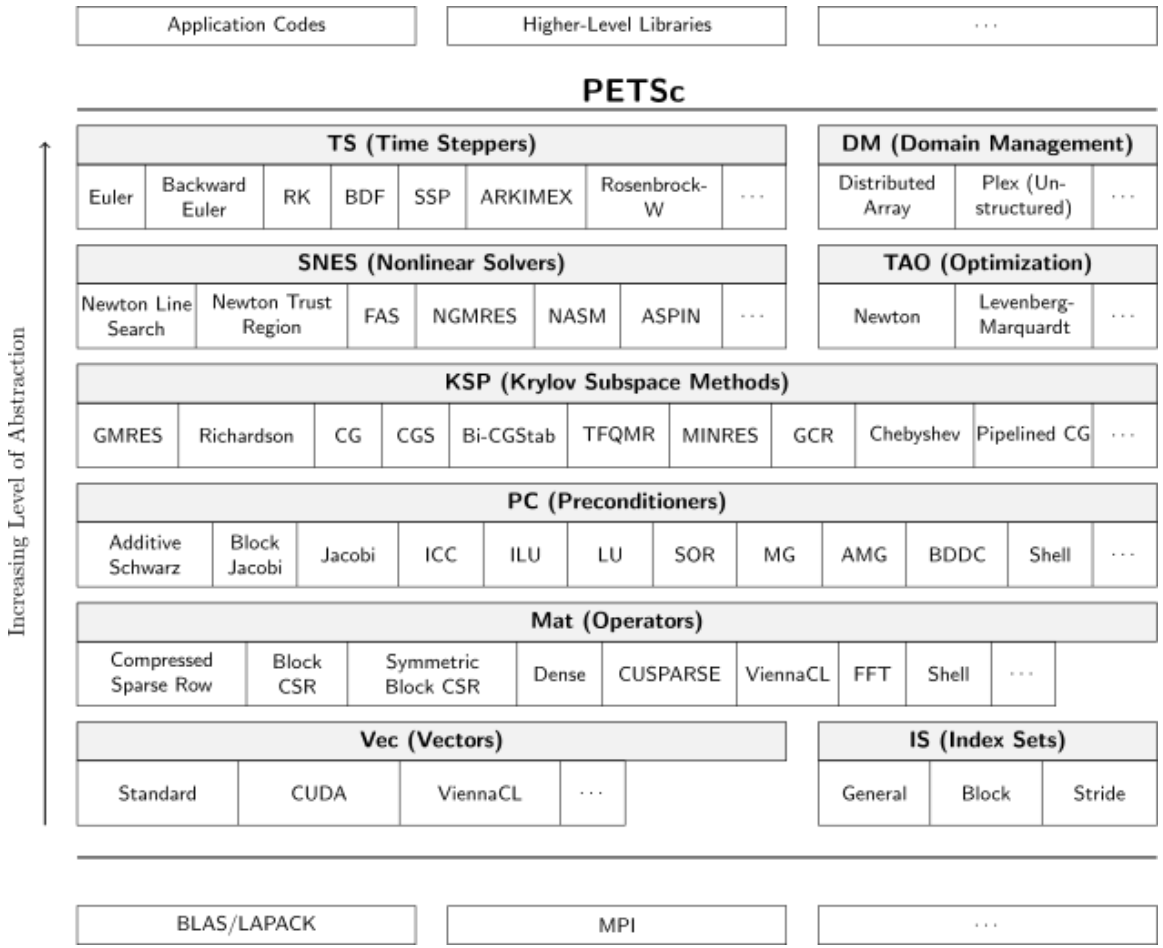
Performance portability with libCEED's matrix-free operators

Extensible Solvers from PETSc

`CeedEvaluator`

`MatCeed`

`CeedVector`



Need to wrap libCEED operators to work with PETSc objects

CeedEvaluator

`CeedEvaluator` wraps a non-linear `CeedOperator`

```
1  CeedEvaluatorCreate(DM dm_x, DM dm_y, CeedOperator op, CeedEvaluator *evaluator);
2  CeedEvaluatorUpdateTimeAndBoundaryValues(CeedEvaluator evaluator, PetscReal time);
3
4  // Different Apply* variants
5  CeedEvaluatorApplyGlobalToGlobal(CeedEvaluator evaluator, Vec X, Vec Y);
6  CeedEvaluatorApplyAddGlobalToGlobal(CeedEvaluator evaluator, Vec X, Vec Y);
7  CeedEvaluatorApplyVelocityGlobalToGlobal(CeedEvaluator evaluator, Vec X, Vec X_dot,
       Vec Y);
8  CeedEvaluatorApplyGlobalToLocal(CeedEvaluator evaluator, Vec X, Vec Y_loc);
9  CeedEvaluatorApplyLocalToLocal(CeedEvaluator evaluator, Vec X_loc, Vec Y_loc);
```

`CeedBoundaryEvaluator` provides essential BCs via `CeedOperator`

```
1  CeedBoundaryEvaluatorCreate(MPI_Comm comm, CeedOperator op_dirichlet,
       CeedBoundaryEvaluator *evaluator);
2
3  // Override DMPlex BC computation
4  DMPlexSetCeedBoundaryEvaluator(DM dm, CeedBoundaryEvaluator evaluator);
```

Material Models

`CeedEvaluator` supports residual and auxiliary comutations

- `TSIFunction`
- `TSI2Function`
- `TSRHSFunction`
- `SNESFunction`
- `SNESObjective`
- Diagnostic/output value computation

MatCEED

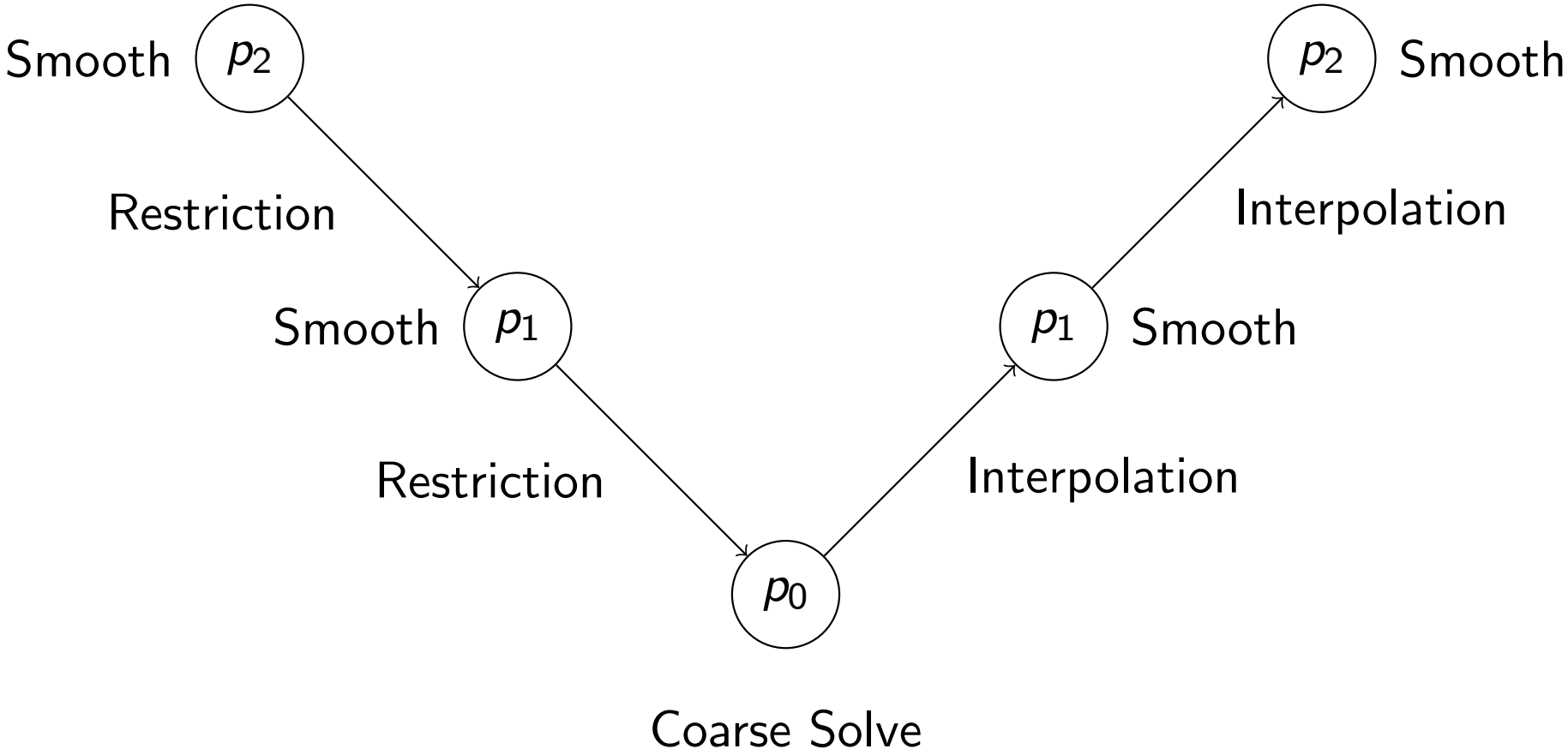
`MatCEED` wraps a linear `CeedOperator`

```
1  MatCeedCreate(DM dm_x, DM dm_y, CeedOperator op_mult, CeedOperator op_mult_transpose,
       Mat *mat);
2  MatCeedSetTime(Mat mat, PetscReal time);
3
4  // COO assembly support
5  MatCeedCreateMatCOO(Mat mat_ceed, Mat *mat_coo);
6  MatCeedSetPreallocationCOO(Mat mat_ceed, Mat mat_coo);
7  MatCeedAssembleCOO(Mat mat_ceed, Mat mat_coo);
8
9  // Private - Core functionality
10 MatMult_Ceed(Mat A, Vec X, Vec Y)
11 MatMultTranspose_Ceed(Mat A, Vec Y, Vec X)
12
13 // Private - PC support
14 MatGetDiagonal_Ceed(Mat A, Vec D);
15 MatGetDiagonalBlock_Ceed(Mat mat_ceed, Mat *mat_block);
16 MatInvertBlockDiagonal_Ceed(Mat mat_ceed, const PetscScalar **values);
17 MatInvertVariableBlockDiagonal_Ceed(Mat mat_ceed, PetscInt num_blocks, const PetscInt
       *block_sizes, PetscScalar *values);
```

Native PC Support

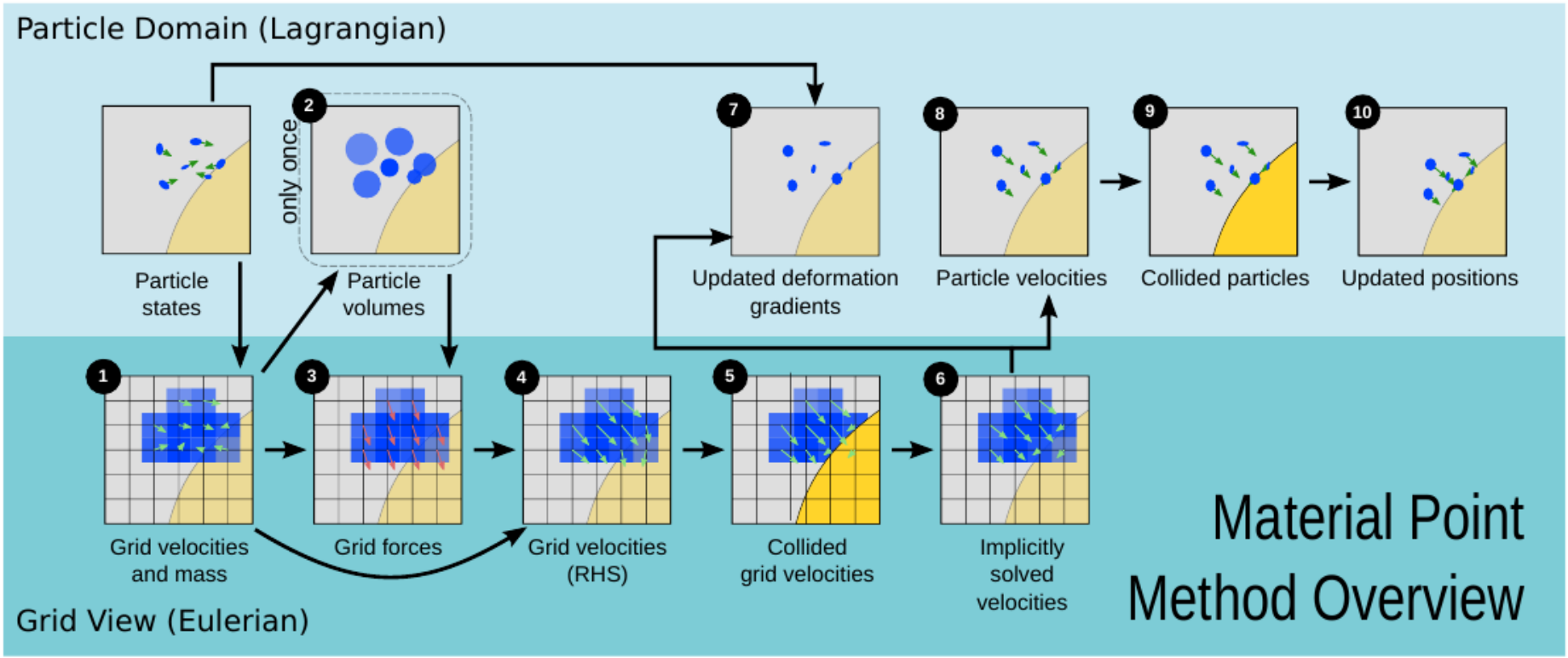
- `PCNone` - :)
- `PCJacobi` - matrix free diagonal assembly
- `PCPBJacobi` - matrix free block diagonal assembly
- `PCVPBJacobi` - matrix free variable block diagonal assembly
- COO assembly into AIJ for all other PCs

PCpMG



MatCEED directly supports all operations except GAMG coarse solve

What is MPM?

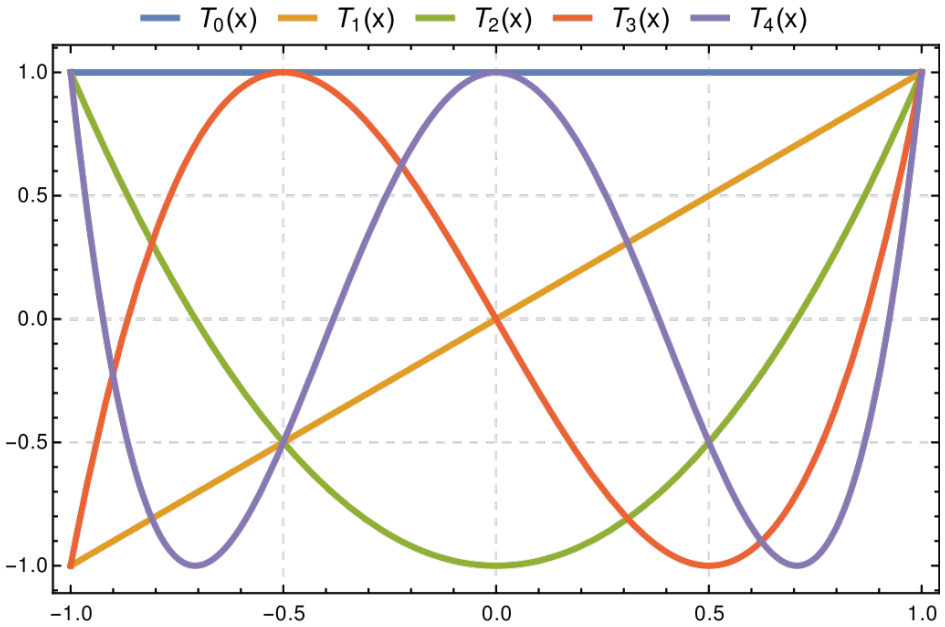


- Continuum based particle method with background mesh for gradients
- Extension of FLIP (which is an extension of PIC)
- Used in rendering for the movie *Frozen*

What does MPM have to do with FEM?

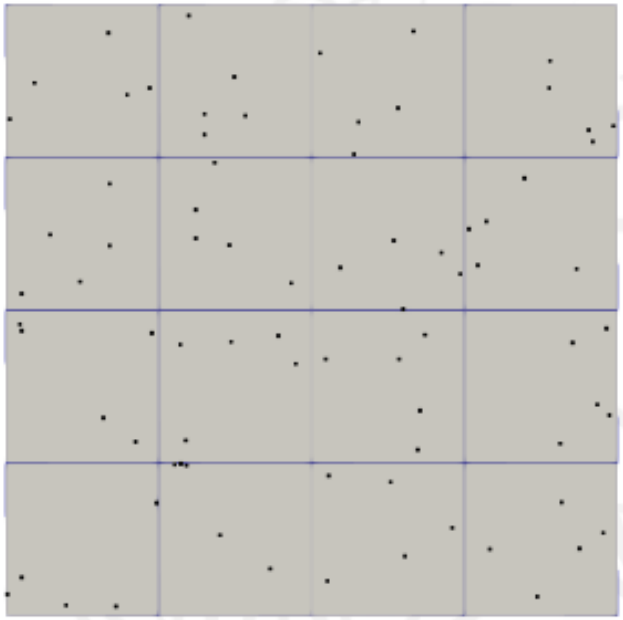
- Problem on background mesh changes when material points move
- Natural fit for matrix-free representation
- Similar reasoning to use matrix-free for adaptive methods
- Ratel FEM infrastructure provides fast background mesh solves

libCEED Basis Evaluation AtPoints



- Interpolate from primal to dual (quadrature) space
- Fit Chebyshev polynomials to values at quadrature points
- Evaluate Chebyshev polynomials at reference coords of material points
- Transpose the order for projection to mesh from material points

DMSwarm for Material Points



- DMSwarm manages material points
- DMPlex manages cells (elements)
- API for cell reference coordinates of points

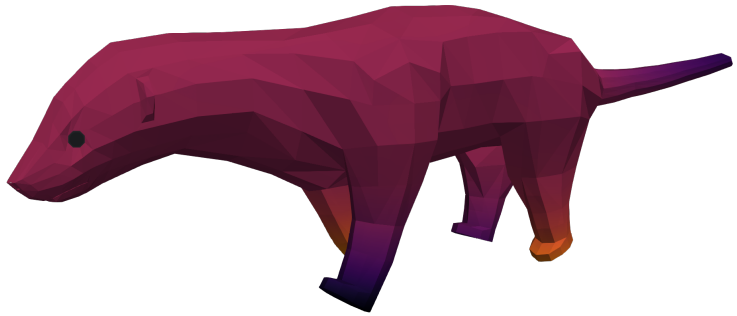
Current MPM Work

- CeedOperator support AtPoints on CPU
- GPU support - ElemRestriction complete, Basis in progress
- Ratel implements MPM for CEED BPs, elasticity in progress (soon)

Future Work

- CeedEvaluator
  - Further testing in Ratel/HONEE desirable
  - Need API for arbitrary number of input/output vectors
- MatCEED
  - Ready for initial upstreaming
  - Wrap inner AIJ for other PCs (GAMG!)
- Develop QFunction specification (resolution independent)
- We invite contributors and friendly users

Questions?



libCEED Repo: https://github.com/CEED/libCEED
Ratel Repo: https://gitlab.com/micromorph/ratel

Grant: Predictive Science Academic Alliance Program (DE-NA0003962)